Spring Cloud Stream (ActiveMQ | RabbitMQ)

实践操作

- 项目中消息队列选型使用的是Kafka
- Spring Cloud Stream对消息队列进行抽象，使使用者对消息队列可以低感知。
- 实践操作 略

笔记

操作ActiveMQ

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合、异步消息、流量削锋等问题，实现高性能、高可用、可伸缩和最终一致性架构，是大型分布式系统不可缺少的中间件。

目前在生产环境中使用较多的消息队列有 ActiveMQ、RabbitMQ、ZeroMQ、Kafka、MetaMQ、RocketMQ

特性

- 异步性：将耗时的同步操作通过以发送消息的方式进行了异步化处理，减少了同步等待的时间。
- 松耦合：消息队列减少了服务之间的耦合性，不同的服务可以通过消息队列进行通信，而不用关心彼此的实现细节，只要定义好消息的格式就行。
- 分布式：通过对消费者的横向扩展，降低了消息队列阻塞的风险，以及单个消费者产生单点故障的可能性（当然消息队列本身也可以做成分布式集群）。
- 可靠性：消息队列一般会把接收到的消息存储到本地硬盘上（当消息被处理完之后，存储信息根据不同的消息队列实现，有可能将其删除），这样即使应用挂掉或者消息队列本身挂掉，消息也能够重新加载。

JMS 规范

JMS 即 Java 消息服务（Java Message Service）应用程序接口，是一个 Java 平台中关于面向

消息中间件（MOM）的 API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。Java 消息服务是一个与具体平台无关的 API，绝大多数 MOM 提供商都对 JMS 提供支持。

JMS 的消息机制有 2 种模型，一种是 Point to Point，表现为队列的形式，发送的消息，只能被一个接收者取走；另一种是 Topic，可以被多个订阅者订阅，类似于群发。

ActiveMQ 是 JMS 的一个实现。

ActiveMQ 介绍

ActiveMQ 是 Apache 软件基金下的一个开源软件，它遵循 JMS1.1 规范（Java Message Service），是消息驱动中间件软件（MOM）。它为企业消息传递提供高可用、出色性能、可扩展、稳定和安全保障。

- 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

- application.properties 中添加配置。

```
# 基于内存的 ActiveMQ
spring.activemq.in-memory=true
# 不适应连接池
spring.activemq.pool.enabled=false

# 独立安装的 ActiveMQ
#spring.activemq.broker-url=tcp://192.168.0.1:61616
#spring.activemq.user=admin
#spring.activemq.password=admin
```

- 队列（Queue） 队列发送的消息，只能被一个消费者接收。

创建队列

```
@Configuration
public class MqConfig {
@Bean
public Queue queue() {
```

```
return new ActiveMQQueue("neo.queue");
}
}
```

使用 @Configuration 注解在项目启动时，定义了一个队列 queue 命名为：neo.queue。

- 消息生产者
  创建一个消息的生产者：

  ```
  @Component
  public class Producer{
  @Autowired
  private JmsMessagingTemplate jmsMessagingTemplate;
  @Autowired
  private Queue queue;
  public void sendQueue(String msg) {
  System.out.println("send queue msg :"+msg);
  this.jmsMessagingTemplate.convertAndSend(this.queue, msg);
  }
  }
  ```

  JmsMessagingTemplate 是 Spring 提供发送消息的工具类，使用 JmsMessagingTemplate 和创建好的 queue 对消息进行发送。

- 消息消费者
  ```
  @Component
  public class Consumer {

  @JmsListener(destination = "neo.queue")
  public void receiveQueue(String text) {
  System.out.println("Consumer queue msg : "+text);
  }
  }
  ```

使用注解 @JmsListener(destination = "neo.queue")，表示此方法监控了名为 neo.queue 的队列。当队列 neo.queue 中有消息发送时会触发此方法的执行，text 为消息内容。

- 广播（Topic） 广播发送的消息，可以被多个消费者接收。

创建 Topic

```
@Configuration
public class MqConfig {
@Bean
public Topic topic() {
return new ActiveMQTopic("neo.topic");
}
}
```

使用 @Configuration 注解在项目启动时，定义了一个广播 Topic 命名为：neo.topic。

- 消息生产者 创建一个消息的生产者：

```
@Component
public class Producer{
    @Autowired
    private JmsMessagingTemplate jmsMessagingTemplate;
    @Autowired
    private Topic topic;
    public void sendTopic(String msg) {
        System.out.println("send topic msg :"+msg);
        this.jmsMessagingTemplate.convertAndSend(this.topic, msg);
    }
}
```

和上面的生产者对比只是 convertAndSend() 方法传入的第一个参数变成了 Topic。

- 消息消费者

```
@Component
public class Consumer {

    @JmsListener(destination = "neo.topic")
    public void receiveTopic(String text) {
        System.out.println("Consumer topic msg : "+text);
    }
}
```

消费者也没有变化，只是监听的名改为上面的 neo.topic，因为模拟多个消费者，复制一份 Consumer 命名为 Consumer2，代码相同在输出中标明来自 Consumer2。

测试
创建 SampleActiveMqTests 测试类，注入创建好的消息生产者。

同时支持队列（Queue）和广播（Topic）
Spring Boot 集成 ActiveMQ 的项目默认只支持队列或者广播中的一种，通过配置项 spring.jms.pub-sub-domain 的值来控制，true 为广播模式，false 为队列模式，默认情况下支持队列模式。

如果需要在同一项目中既支持队列模式也支持广播模式，可以通过 DefaultJmsListenerContainerFactory 创建自定义的 JmsListenerContainerFactory 实例，之后在 @JmsListener 注解中通过 containerFactory 属性引用它。

分别创建两个自定义的 JmsListenerContainerFactory 实例，通过 pubSubDomain 来控制是支持队列模式还是广播模式。

```
@Configuration
@EnableJms
public class ActiveMQConfig {

    @Bean("queueListenerFactory")
    public JmsListenerContainerFactory<?> queueListenerFactory(ConnectionFactory co
nnectionFactory) {
        DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContaine
rFactory();
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(false);
        return factory;
    }

    @Bean("topicListenerFactory")
    public JmsListenerContainerFactory<?> topicListenerFactory(ConnectionFactory co
nnectionFactory) {
        DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContaine
rFactory();
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(true);
        return factory;
    }
}
```

然后在消费者接收的方法中，指明使用 containerFactory 接收消息。
```
@Component
public class Consumer {
```

```
@JmsListener(destination = "neo.queue", containerFactory = "queueListenerFactory")
public void receiveQueue(String text) {
    System.out.println("Consumer queue msg : "+text);
}

@JmsListener(destination = "neo.topic", containerFactory = "topicListenerFactory")
public void receiveTopic(String text) {
    System.out.println("Consumer topic msg : "+text);
}
```

}
```

改造完成之后，再次执行队列和广播的测试方法，就会发现项目同时支持了两种类型的消息收发。

# RabbitMQ

## RabbitMQ 介绍

AMQP（Advanced Message Queuing Protocol，高级消息队列协议）是应用层协议的一个开放标准，为面向消息的中间件设计。消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然。

AMQP 的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。

RabbitMQ 是一个开源的 AMQP 实现，服务器端用 Erlang 语言编写，支持多种客户端，
```